Interactive Ray Tracing

Steven Parker William Martin Peter-Pike Sloan
Peter Shirley Brian Smits Charles Hansen
Computer Science Department
University of Utah

Interactive Ray Tracing

- SGI Origin 2000
- 64 processors
- Display is only graphics hardware used

- Video recorded directly from screen
- 600 x 437 resolution

Why is this fast?

- Ray tracing performs well on modern processors
- For static scenes, runtime grows slower than number of objects rendered
- Parallelism

What we didn't do

- Reuse of information (from previous frames)
- Interpolation between pixels
- Explicitly optimized code (all C++)
- Complex load balancing
- Scan conversion (hardware or software)

Guiding principles

- KISS programs are good
- Careful attention to data locality is essential
- Careful attention to counting flops is not essential
  - Most things are re-computed instead of stored

Serial Efficiency

- Judicious use of C++ features
- Memory locality
- Minimizing expensive operations (sqrt, divide)

- Approximately three hours of optimizing for each hour of coding

Parallel Efficiency

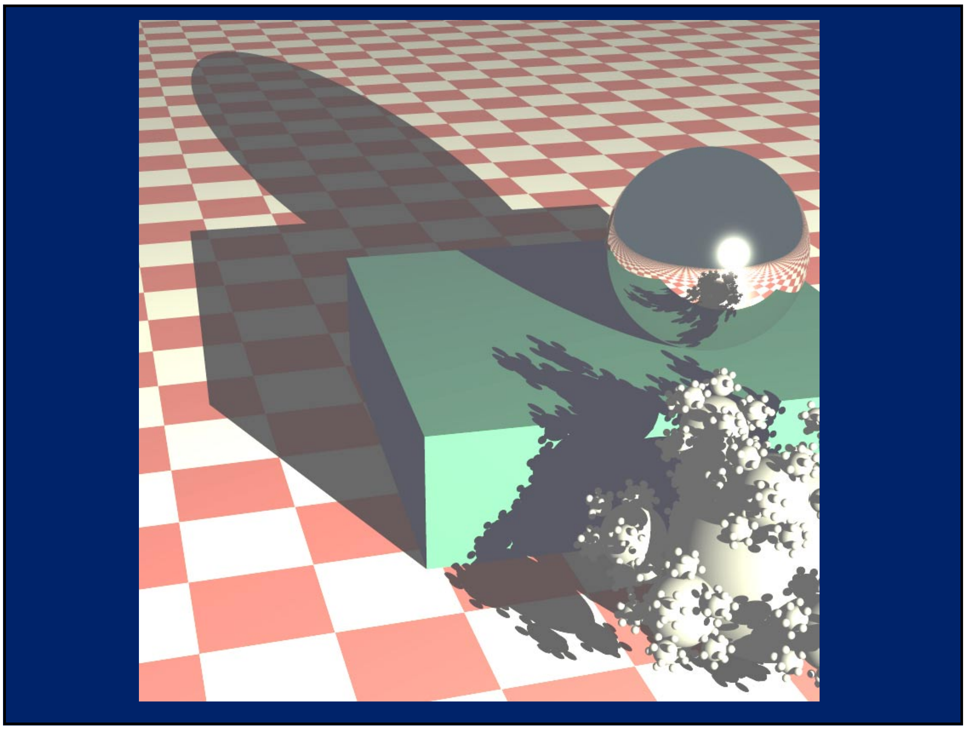
- Dynamic load balancing
- Use Origin fetch&op counter

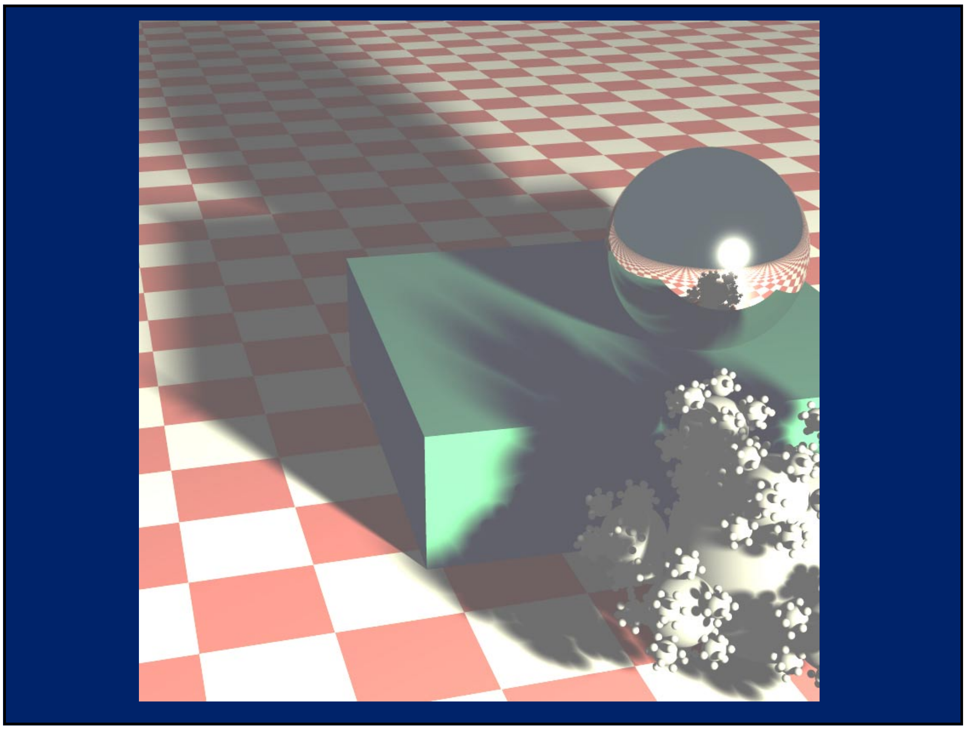
- Straightforward implementation
- Not tuned to topology of underlying architecture (bristled hypercube)

New Ray Tracing Mentality

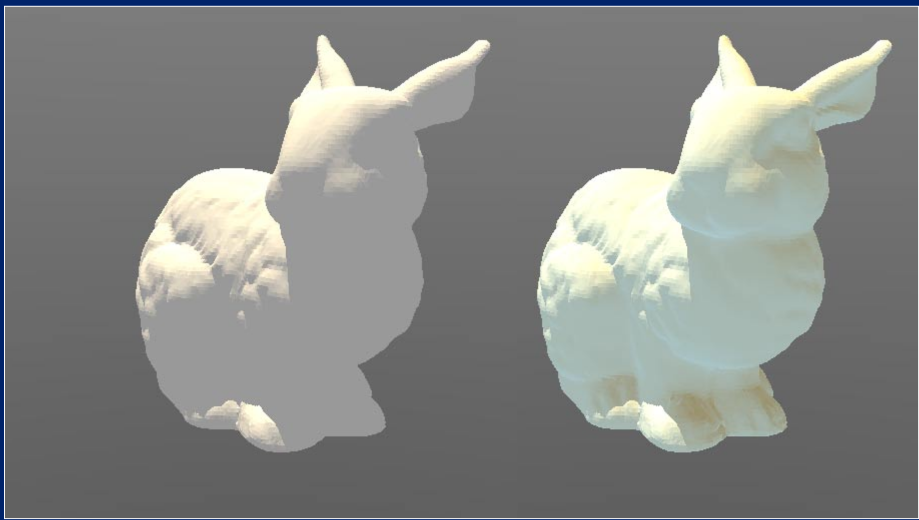
- How can one achieve important visual cues without impacting interactivity?

- Soft shadows
- Directionally varying ambient term





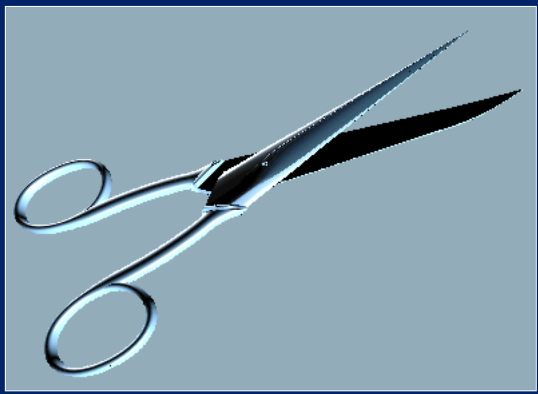
Ambient Lighting



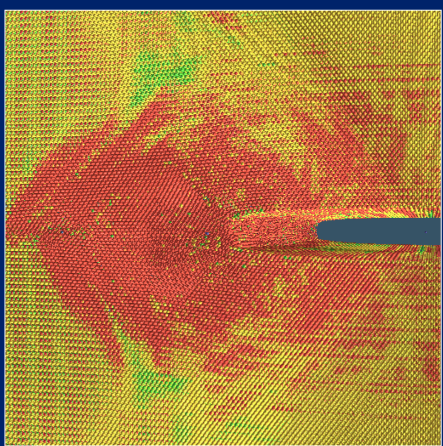
Rich Primitives

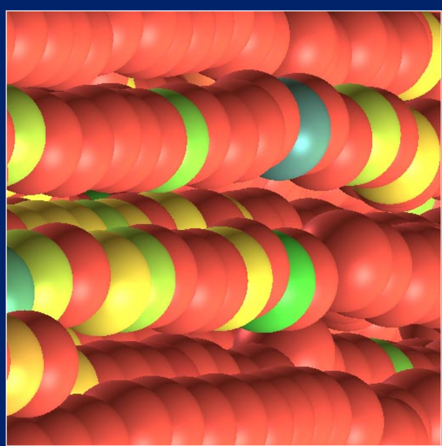
- Ray tracing can accommodate very large and complex data
- Adding complex primitives is just as easy as in a batch ray tracer

Spline models



35 million spheres



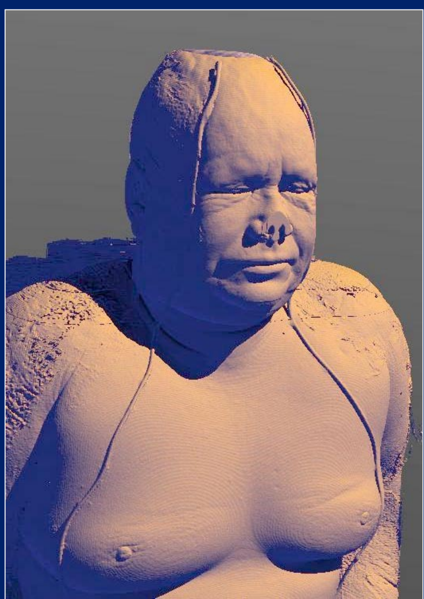


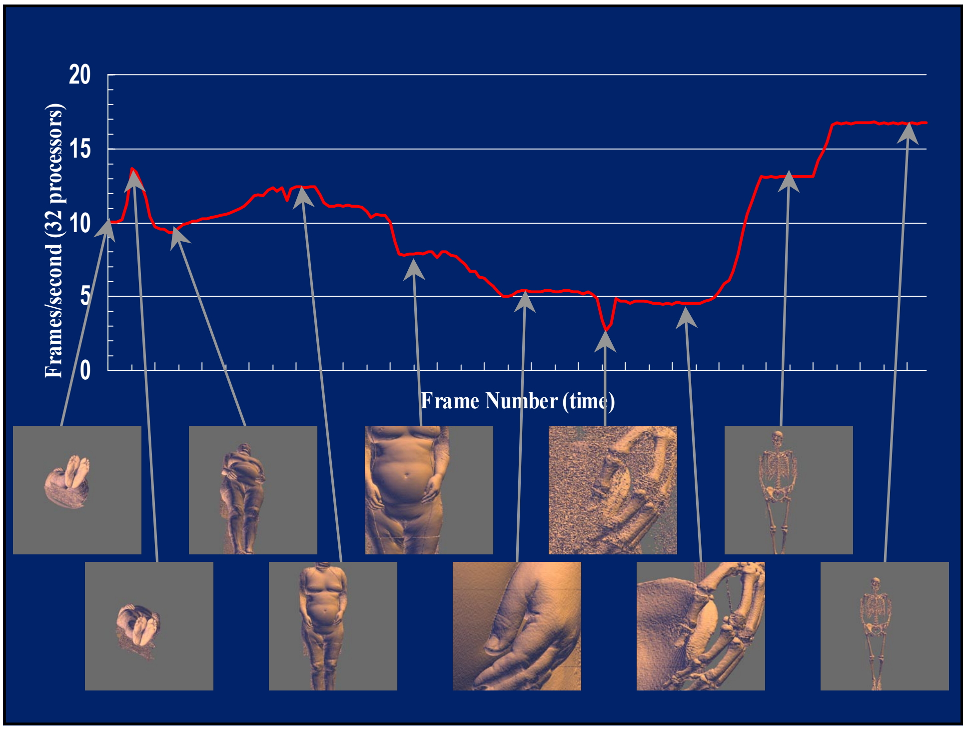
Textured Volume Data

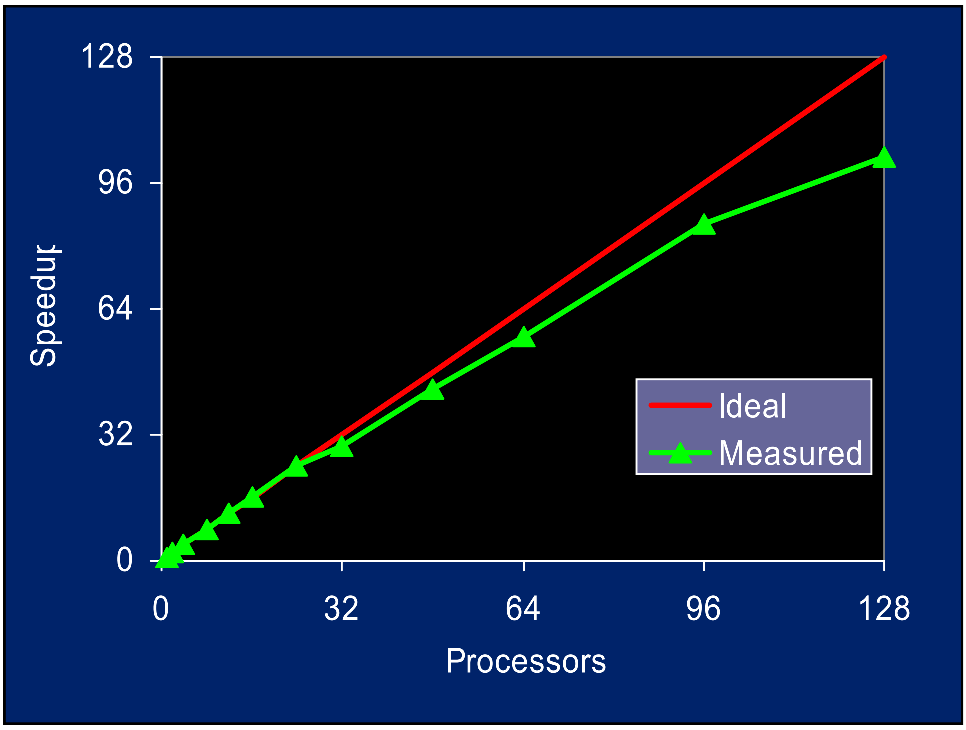
Maximum Intensity Projection

Performance

- Rendering of isosurfaces from visible female CT dataset (900 Megabytes)
- More details of this technique in Visualization ‘98 paper




Frames/second (32 processors)
20
15
10
5
0
Frame Number (time)


128
96
64
32
0
Speedup
0
32
64
96
128
Processors
Ideal
Measured

Efficiency of Data Access

For visible female:

L1 cache hits: 99.44%

L2 cache hits: 97.6%

Memory bandwidth: 2.1 MB/sec/processor

Teapot scene: 8 MB/sec/processor

Frameless Rendering

- Improves interactivity
- Lowers memory locality
- Relaxes synchronization

- Helpful if off by a factor of 5, but not by a factor of 20

Interactive Ray Tracer

- Useful tool for interactively exploring complex scenes on large machines
- Good research tool for prototyping
- Attention to memory system critical for performance

Problems with current system

- Some scenes and algorithms just too slow
- Preprocessing precludes dynamic scenes
- No Antialiasing
- Variable frame rate

Planned Improvements

- New API for scene graph ray tracing
- Dynamic efficiency structures that amortize overhead cost
- Parallel front end for pixel reprojection
- 1000+ processor implementation

Futurism

- Good research tool now, but will it ever play video games?
- Obviates many graphics processor bottlenecks, but also introduces new ones

Future 1: Better Hardware

- Moore’s Law-- in ten years CPUs will be 100x faster with 10x memory bandwidth
- Current system uses only 10% of memory bandwidth
- Will likely still perform well in ten years

- Custom hardware?

Future 2: More CPUs

- Los Alamos cluster has 48 128CPU O2Ks with approximately 125x the raw power of our current machine
- Bandwidth to frame buffer would allow 40 uncompressed HDTV images per second to be ray traced
- Not yet practical for the desktop

ASCI
Blue Mountain

Future 3: Better reuse of computation

- Pixels can be reprojected between frames
- New pixels are traced as needed

Video

Interactive Rendering using the Render Cache

Bruce Walter (iMAGIS)
George Drettakis (iMAGIS)
Steven Parker (Univ. of Utah)

Future 4: Hybrid

- Better CPU's - just wait
- More CPU's - just get more money
- More intelligence - gotta work

Evangelism

- This isn't hard
- This is fun
  - A good prototyping tool
- Necessary hardware is becoming affordable for research institutions

Overview

- Isosurfacing is performed implicitly at every pixel
- Maps well onto modern architectures
- Interactive for some datasets on some machines

Video

- SGI Origin 2000 using 50 processors
- 512 x 512 image
- 512 x 512 x 1734 voxels (900 Megabytes)

  Visible Female data from the National Library of Medicine Visible Human Project

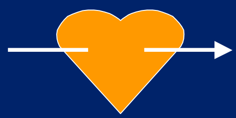
Outline

I. Ray tracing isosurfaces

II. Achieving interactivity

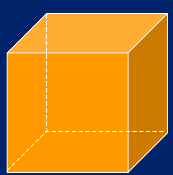
Isosurfacing for Analytic Functions

- f(x,y,z)=0
- ray tracing via root finding (e.g. Kalra and Barr ‘89)



- explicit polygonalization (e.g. Stander and Hart ‘97)

Trilinear Cells are Easier



$$
\begin{array}{rccccc}
\rho(u,v,w) = & (1-u) & (1-v) & (1-w) & \rho_{000} & + \\
 & (1-u) & (1-v) & w & \rho_{001} & + \\
 & (1-u) & v & (1-w) & \rho_{010} & + \\
 & (1-u) & v & w & \rho_{011} & + \\
 & u & (1-v) & (1-w) & \rho_{100} & + \\
 & u & (1-v) & w & \rho_{101} & + \\
 & u & v & (1-w) & \rho_{110} & + \\
 & u & v & w & \rho_{111} &
\end{array}
$$

Isosurfacing for a Trilinear Cell

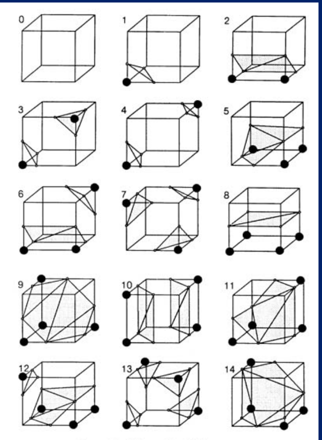


Figure 3. Triangulated Cubes.

Marching Cubes

Lorensen and Cline ('87)

Wyvill and Wyvill ('86)

Why Not Always Use Marching Cubes?

Marching cubes can generate millions of polygons

- Reduce by decimation (e.g. Shekhar et. al '96)
- Reduce by culling (e.g. Livnat and Hansen '98)

Isosurfacing for a Trilinear Cell

cell:

$$
\begin{aligned}
\rho(u,v,w) = {} & (1-u)\,(1-v)\,(1-w)\,\rho_{000} + \\
& (1-u)\,(1-v)\,w\,\rho_{001} + \\
& (1-u)\,v\,(1-w)\,\rho_{010} + \\
& (1-u)\,v\,w\,\rho_{011} + \\
& u\,(1-v)\,(1-w)\,\rho_{100} + \\
& u\,(1-v)\,w\,\rho_{101} + \\
& u\,v\,(1-w)\,\rho_{110} + \\
& u\,v\,w\,\rho_{111}
\end{aligned}
$$

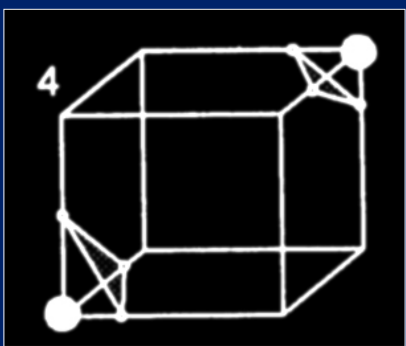
ray:

$$
\begin{aligned}
u &= u_0 + tu_1 \\
v &= v_0 + tv_1 \\
w &= w_0 + tw_1
\end{aligned}
$$

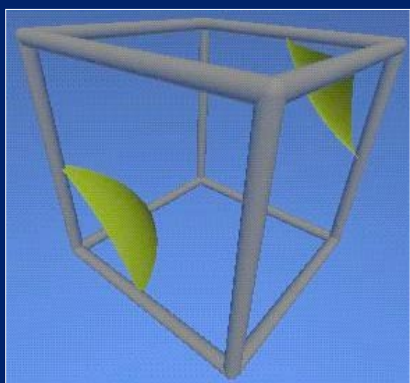
ray intersects cell where:

$$At^3 + Bt^2 + Ct + D = 0$$

Isosurfacing for a Piecewise Linear Cell

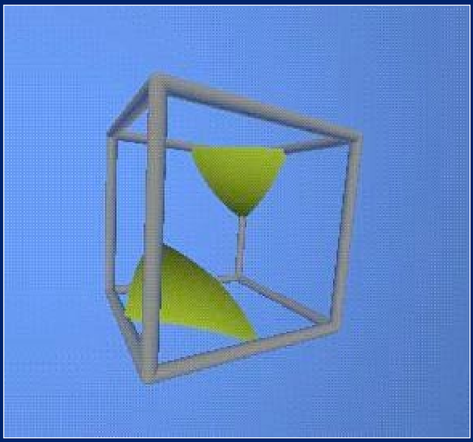


marching cubes

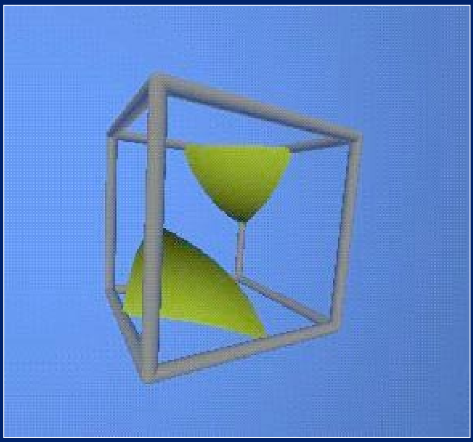


ray tracing

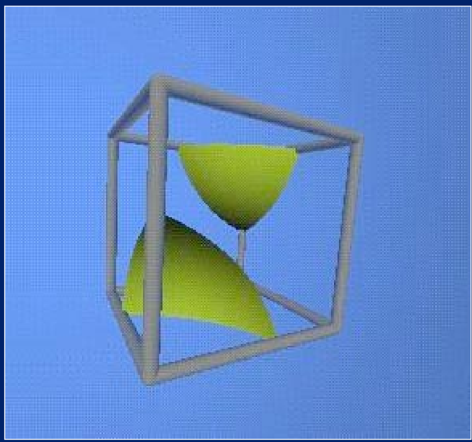
Effects of Direct Cubic Solution



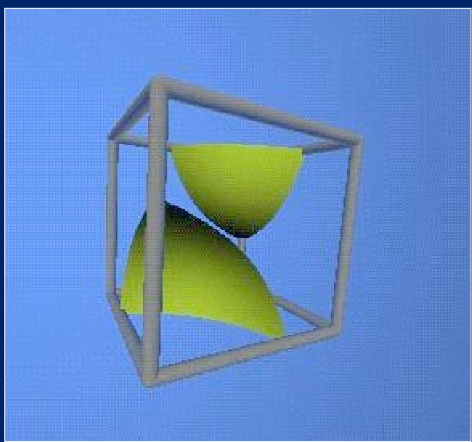
Effects of Direct Cubic Solution



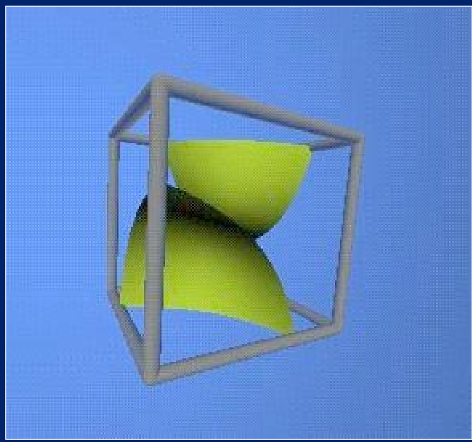
Effects of Direct Cubic Solution



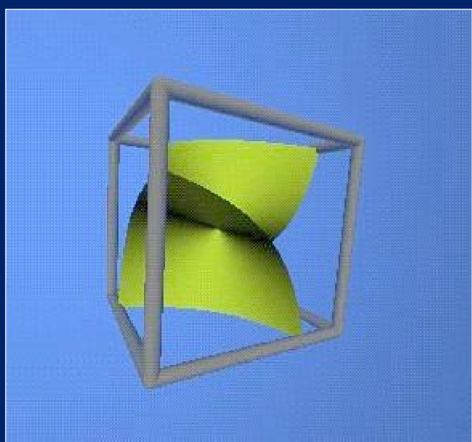
Effects of Direct Cubic Solution



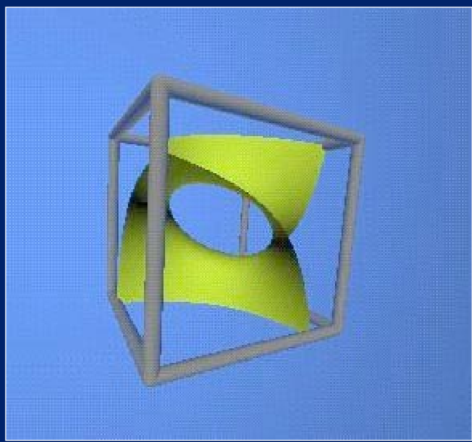
Effects of Direct Cubic Solution



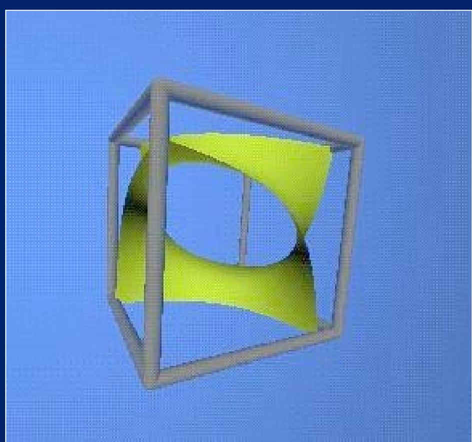
Effects of Direct Cubic Solution



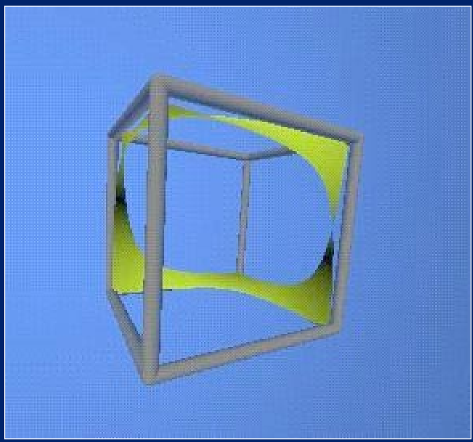
Effects of Direct Cubic Solution



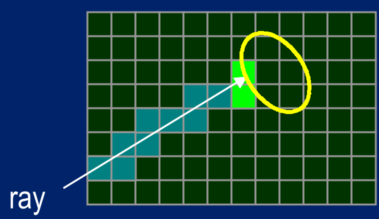
Effects of Direct Cubic Solution

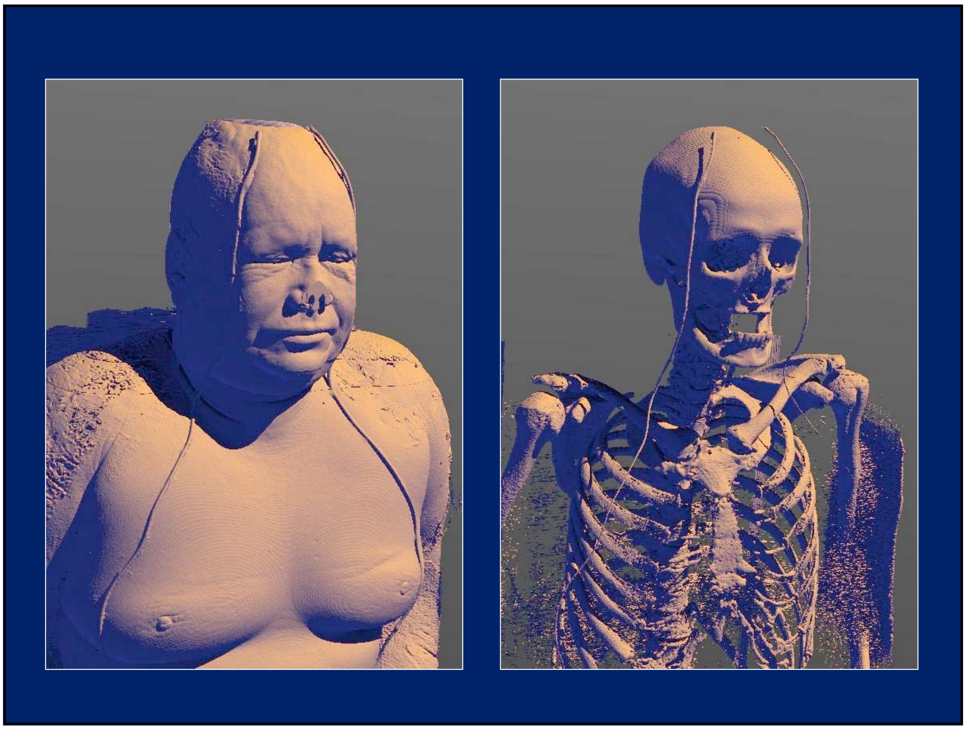


Effects of Direct Cubic Solution



Isosurfacing for a grid of cells





Previous Ray Tracing for Isosurfaces

- Marschner and Lobb ('94)
- Lin and Ching ('96)

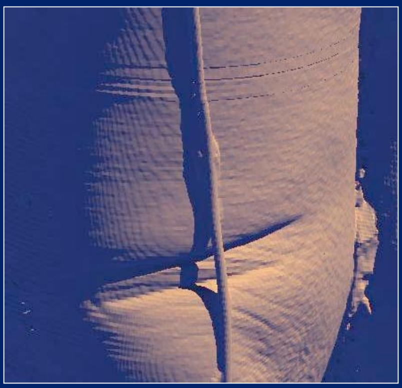
Feature Comparison

Ray Tracing
- Implicit geometry
- Software shading

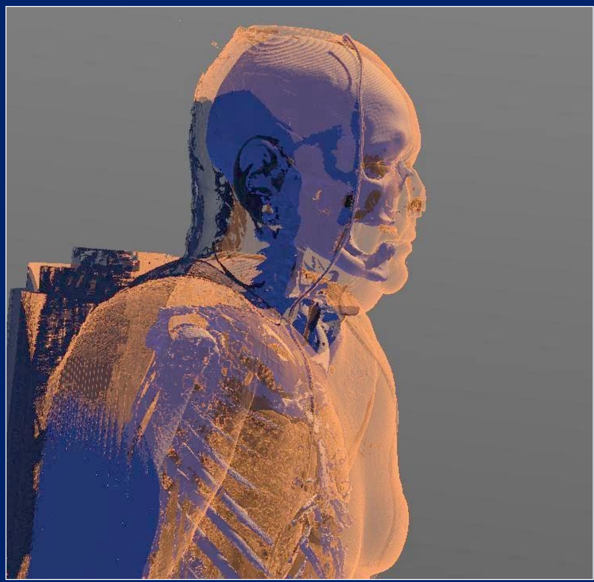
Marching Cubes
- Explicit geometry
- Hardware shading

Shadows

without



with



Performance Comparison

Ray Tracing
- Run time proportional to image size
- Highly scalable

Marching Cubes
- Run time proportional to data size
- Leverages conventional graphics hardware

How Fast is Ray Tracing?

- A single R10000 (195 Mhz)
- 512 x 512 image
- 512 x 512 x 1734 voxels (900 Megabytes)
  Visible Female data from the National Library of Medicine Visible Human Project
- Times vary from 22 to 418 seconds per frame

Optimizations

- Parallelism
- Hierarchical data representation
- Data layout for better locality

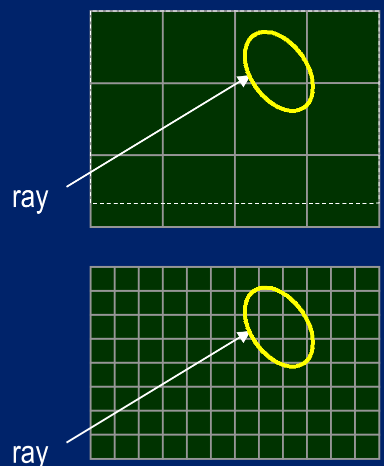
Parallel Implementation

- Implemented on SGI Origin 2000 ccNUMA architecture - up to 128 processors
- Approximately linear speedup
- Load balancing and memory coherence are keys to performance

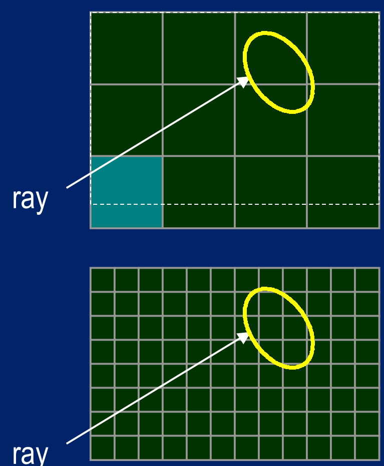
Hierarchical Data Representation

- Skip over cells which do not contain an isosurface - Wilhelms and van Gelder ('90)
- Keep "macrocells" which contain the min/max values for contained cells

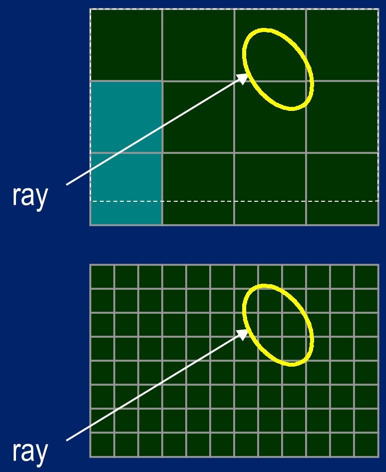
Two Level Representation



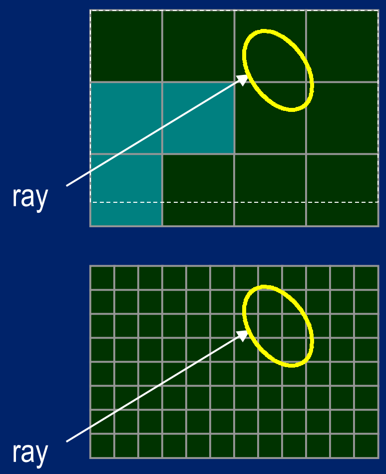
Two Level Representation



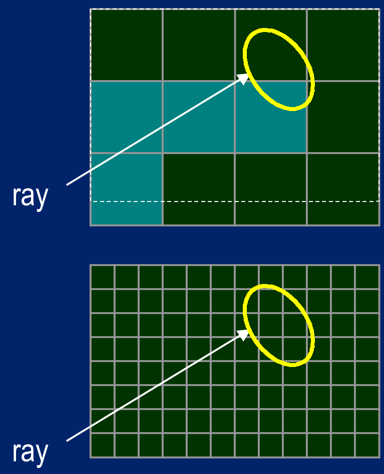
Two Level Representation



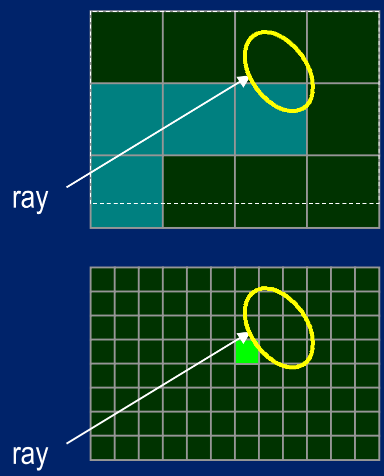
Two Level Representation



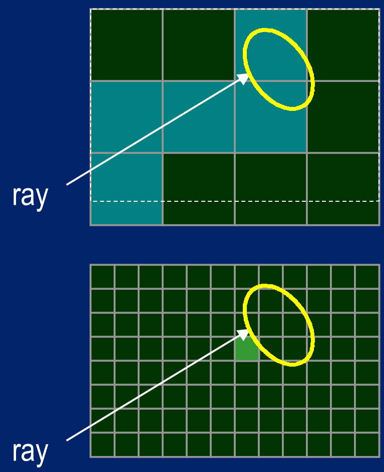
Two Level Representation



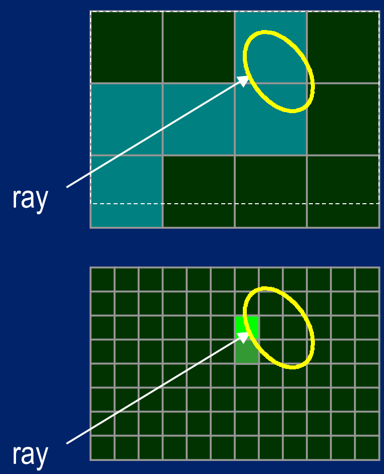
Two Level Representation



Two Level Representation



Two Level Representation



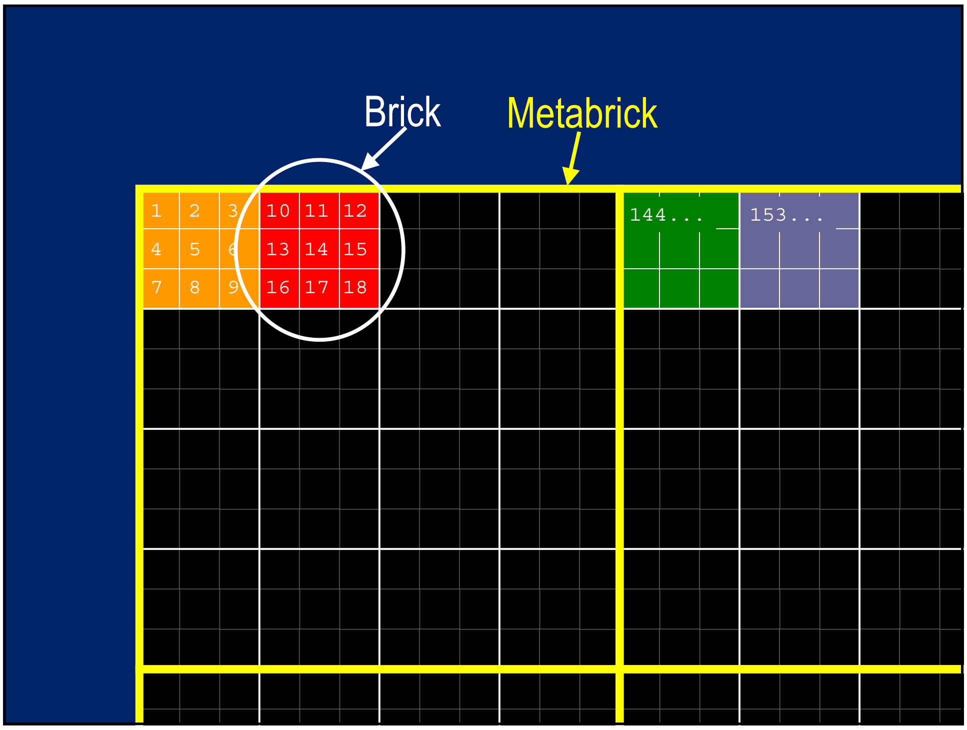
Number of Hierarchy Levels

- Traversal from cell to cell is cheaper than moving up and down levels
- Would like to skip large empty regions
- We use 3 or 4 levels in practice

Data Layout (Bricking)

- Optimizing for memory locality
- Two levels (bricks and metabricks)

- Common trick (e.g. Cox and Ellsworth ‘97)



Data Layout (Bricking)

- Brick sizes (Cache line and page sized cubes)
  - 16 bit data: 5^3 metabricks of 4^3 bricks
  - 32 bit data: 6^3 metabricks of 3^3 bricks

Combining Hierarchy and Bricking

- Requirements of hierarchy are different than the brick sizes
- Traversal at finest level of hierarchy can cross brick boundaries
- Must compute indices into bricked array

Indexing

- Consider 6x6x6 bricks of 3x3x3 bricks:

```
index = (x/3/6)*6*6*6*3*3*3*ny*nz +
  (y/3/6)*6*6*6*3*3*3*nz +
  (z/3/6)*6*6*6*3*3*3 + (x/3%6)*6*6*3*3*3
  + (y/3%6)*6*3*3*3 + (z/3%6)*3*3*3 +
  (x%3)*3*3 + (y%3)*3 + (z%3)
```

- Very expensive
  - Integer division and modulus

What about that function?

```
index = (x/3/6)*6*6*6*3*3*3*ny*nz +
  (y/3/6)*6*6*6*3*3*3*nz +
  (z/3/6)*6*6*6*3*3*3 + (x/3%6)*6*6*3*3*3
  + (y/3%6)*6*3*3*3 + (z/3%6)*3*3*3 +
  (x%3)*3*3 + (y%3)*3 + (z%3)
```

index = $f_x(x) + f_y(y) + f_z(z)$

Efficiency of Data Access

For isosurfacing, lookup 6 index values for 8 data value lookups (instead of 24)

L1 cache hits: 99.44%

L2 cache hits: 97.6%

Memory bandwidth: 2.1 MB/sec/processor

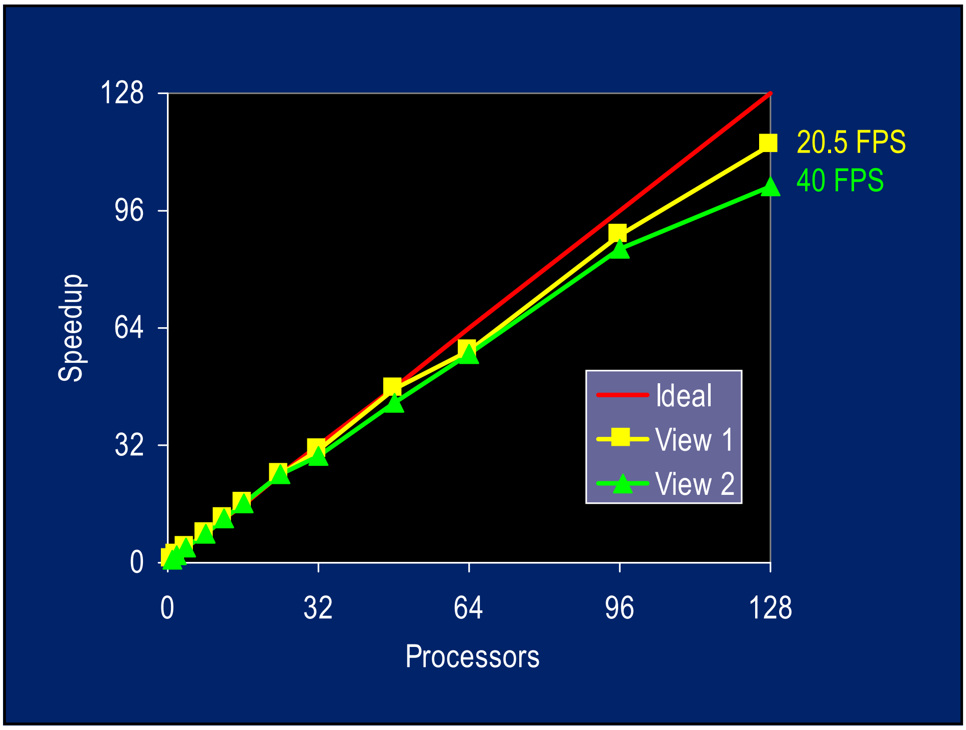
Optimization Results

View	Initial	Bricking	Hierarchy + Bricking
Skin: front	22.4	20.8	8.5
Bone: front	38.4	33.6	8.3
Bone: close	57.6	56.0	12.2
Bone: from feet	417.6	92.8	9.9

Times in seconds for a 512 x 512 image on 1 processor

Where time is spent

Isosurface	Traversal	Intersection	Shading
Skin	55%	22%	23%
Bone	66%	21%	13%



Results

- Gigabyte dataset (1734x512x512)
- 8-15 Frames per second on 64 processors
- Compare to Marching Cubes:
  - bone isosurface: 9.9 million triangles
  - skin isosurface: 6.7 million triangles

Summary

- Useful tool for interactively exploring large datasets on large machines
- Is complementary to marching cubes
- Attention to machine architecture critical to performance

Future Work

- Application to unstructured data
- Frameless rendering
- Ray tracing for other types of scientific data (streamlines, slices, others?)
- Time varying data (> main memory)
- Higher order interpolation methods
- Distributed implementation

Thanks to:

- Richard Coffey, SCI Group, SGI Support
- Jamie Painter at the Advanced Computing Laboratory, Los Alamos National Laboratory
- DOE ASCI and AVTC
- NSF
- Utah State Centers of Excellence
- SGI Visual Supercomputing Center